Resource Aware Adaptive Applications: the CHAMELEON approach

Paolo Di Benedetto
University of L’Aquila (Italy)
paolo.dibenedetto@di.univaq.it

Roadmap

- Introduction
- The CHAMELEON Framework
  - Development Environment
  - Resource Model
  - Analyzer
  - Customizer
- Considerations
- Conclusions and Future Work
Introduction 1/2

- **Ubiquitous networking** allows mobile users to access software services across heterogeneous infrastructures by (resource-constrained) mobile devices.

- To offer users the best Quality according to their needs and specific execution environment, applications need to be:
  - **Context-aware**: to be aware of user needs and resources offered by the execution environment
  - **Adaptive**: the capability of changing applications in order to comply with the current context condition

Introduction 2/2

Approaches to adaptable applications:

- **Self-Contained**: embed the adaptation logic as a part of the application
  - Inherently dynamic, automatically adapt at runtime changes
  - Overhead imposed by the adaptation logic; size of code

- **Tailored**: are the result of an adaptation process which has been previously applied on a generic version of the application
  - Light and optimized code suitable also for limited devices
  - Adaptation is limited at deployment time
Our contribution

CHAMELEON

- Framework for tailoring adaptable applications
- Based on a declarative and deductive approach:
  - permits the specification of "generic" adaptable application code,
  - determines the best adaptation w.r.t. a given execution context,
  - deliver it via OTA.
- Adaptation at deployment time
- Self-evolution based on un-deployment and re-deployment of tailored applications

The CHAMELEON Framework
Development Environment

CHAMELEON Programming Model

Adaptable public class e-learningMidlet extends PMmidlet {
    State state;
    public e-learningMidlet() {...}
    "life-cycle management methods"
    void pauseApp() {...}
    void startApp() {...}
    void destroyApp() {...}
    "e-learning specific state management methods"
    adaptable void saveState();
    adaptable void restoreState();
    "e-learning specific methods"
    adaptable void connect();
    adaptable void getLesson();...
}

Adaptable public class SPRS adapts e-learningMidlet {
    void connect() {
        Annotation.slsAnnotation("Cost(low),e-lQuality(low)");
        plasticMiddleware.connectViaSPRS();
    }
    void getLesson() {"stream the lesson slides"}
    void saveState() {"save the number of the current slide"}
    void restoreState() {"restore the saved state"}
}

Adaptable public class WiFi adapts e-learningMidlet {
    void connect() {
        Annotation.slsAnnotation("Cost(high),e-lQuality(high)");
        plasticMiddleware.connectViaWiFi();
    }
    void getLesson() {"stream the video lesson with its multimedia contents"
        Annotation.resourceAnnotation("Battery(high)");
        ...
    }
    void saveState() {"save the current frame of the video less."}
    void restoreState() {"restore the saved state"}
}
**CHAMELEON Programming Model**

```java
public class C {
    void m1() {...}
    adaptable void a();
    adaptable void b();
}

alternative C1 adapts C {
    void a() {...}
    void b() {...}
}

alternative A1 adapts C {
    void a() {...}
}

alternative A2 adapts C {
    void a() {...}
}

alternative B1 adapts C {
    void b() {...}
}

alternative B2 adapts C {
    void b() {...}
}
```

**CHAMELEON IDE Eclipse plug-in**

- Helps developing adaptable applications
- Is integrated with the Preprocessor
Resource Model

- Resource Model: formal model for characterizing resources
- **Resource**: item/entity required to accomplish an activity/task.
- RM defines Resources as typed identifiers:
  - **Natural** for consumable resources (Battery, CPU, ...)
  - **Boolean** for non consumable resources that can be present or not (API, network radio interface, ...)
  - **Enumerated** for non consumable resources that admits a limited set of values (screen resolution, ...)

```plaintext
define Energy as Natural
define CPU as Natural
define Bluetooth as Boolean
define Resolution as {low, medium, high}
```

Resource Definition
Resource Model

Resource Instance
- Association resource(value)
  - e.g. Bluetooth(true), CPU(1)

Ordering
- Natural: Natural ordering (e.g. CPU(1)<CPU(2))
- Boolean: false < true (Bluetooth(false)<Bluetooth(true))
- Enumerated: position wise (Resolution(low)<Resolution(high))

RM: Resource Instances Operations

- Sum operation
  \[ r(v_1) \oplus r(v_2) = \begin{cases} 
  r(v_1 + v_2) & \text{if typeof}(r) = \text{Natural} \\
  r(\max(v_1, v_2)) & \text{otherwise} 
\end{cases} \]
  - CPU(1) ⊕ CPU(2) = CPU(3)
  - Bluetooth(false) ⊕ Bluetooth(true) = Bluetooth(true)
RM: Resource Instances Operations

- Scalar multiplication operation

\[ n \otimes r(v) = \begin{cases} r(n \times v) & \text{if } \text{typeof}(r) = \text{Natural} \\ r(v) & \text{otherwise} \end{cases} \]

- 3 \otimes CPU(1) = CPU(3)
- 3 \otimes Resolution(low) = Resolution(low)

RM: Resource Sets

Resource Set
- a set of resource instances, with no resource occurring more than once

Resource Sets are used to specify
- Resource Demand: \{Bluetooth(true), Resolution(high)\}
- Resource Supply: \{Bluetooth(true), Resolution(low), Energy(30)\}
RM: Resource Sets Operations

Resource Sets Sum Operations

\[ R_1 \oplus R_2 = S_1 \cup S_2 \]

where

\[ S_1 = \{ r(x) \oplus r(y) \mid r(x) \in R_1, r(y) \in R_2 \} \]

\[ S_2 = (R_1 \cup R_2) \setminus (R_1 \cap R_2) \]

\[
\{\text{CPU(1), Bluetooth(false), Resolution(high)}\} \oplus \\
\{\text{CPU(2), Bluetooth(true)}\} = \\
\{\text{CPU(3), Bluetooth(true), Resolution(high)}\}
\]

RM: Resource Sets Operations

Resource Sets Scalar Multiplication Operation

\[ n \otimes R = \{ n \otimes r \mid \forall r \in R \} \]

\[
3 \otimes \{\text{CPU(1), Bluetooth(false), Resolution(high)}\} = \\
\{\text{CPU(3), Bluetooth(false), Resolution(high)}\}
\]
RM: Resource Sets Compatibility

A resource set (demand) P is compatible with a resource set (supply) Q ($P \preceq Q$) if:

1. (Availability) For every resource instance $r(x) \in P$ there exist a resource instance $r(y) \in Q$.
2. (Wealth) For every pair of resource instances $r(x) \in P$ and $r(y) \in Q$, $p(x) \leq p(y)$.

used to determine if an application can run safely on the execution environment
RM: Resource Set Goodness

- used to choose the best compatible application alternative w.r.t. a given execution context
- based on a notion of priority (P) among resources that expresses the “importance” given to a particular resource consumption
- P: Resources \( \rightarrow \) Integer.
  - \( P(r) < 0 \) → the less \( r \) is consumed the better is (e.g., Energy).
  - \( P(r) = 0 \) → the consumption of resource \( r \) is in influential (CPU)
  - \( P(r) > 1 \) → the more \( r \) is consumed the better is (e.g., Thread)

\[
\text{goodness} = \sum_{r \in \text{Resources}} \frac{P(r)}{\text{Supply}(r)} \cdot \text{Demand}(r)
\]

Supply = \{Bluetooth(true), WiFi(true), Energy(40), CPU(50)\}

Demand_1 = \{Bluetooth(true), Energy(10), CPU(50)\}
Demand_2 = \{WiFi(true), Energy(20), CPU(10)\}

\( \Rightarrow \) Demand_1 and Demand_2 are compatible with Supply

Goodness(Demand_1) = 0.5
Goodness(Demand_2) = -2

\( \Rightarrow \) Demand_1 is better than Demand_2
The Analyzer

Analyzer

- Interpreter that abstracts a standard JVM
- Statically analyzes an application inspecting all the possible computation paths and determines its Resource Demand (resources required to correctly execute the application)
- Worst case analysis based on the resource consumption profile
Analyzer: Resource consumption profiles

- provides the description of the characteristics of a specific execution environment
- specifies the impact that Java bytecode instructions (patterns) have on resources

1) istore_1 → {CPU(2)}
2) invoke.* → {CPU(4)}
3) .* → {CPU(1), Energy(1)}
4) invokevirtual LocalDevice.getLocalDevice() →
   {Bluetooth(true), Energy(20)}

Analyzer: Resource consumption profiles

- can be created on the basis of:
  - experimental results based on benchmarking tools
  - information provided by device manufacturers, network sensors, 
    ....

- Always exists a default RCP, possibly refined w.r.t. the execution environment

- the more the resource profile is detailed, the more the resulting resource demand will be accurate
Analyzer: Resource consumption profiles

resBinding : ByteCode → ResourceSet

1) istore_1 → {CPU(2)}
2) invoke.* → {CPU(4)}
3) .* → {CPU(1), Energy(1)}
4) invokestatic LocalDevice.getLocalDevice() →
   {Bluetooth(true), Energy(20)}

resBinding(istore_1, istore_1)
resBinding(istore_1, istore_1)
resBinding(istore_1, istore_1)

Programming Model: Annotations

- Are used to attach additional information to the generic code to make more precise the analysis

- Permits to specify:
  - upper bound on the number of:
    - loop iterations (Loop Annotation)
    - recursive method calls (Call Annotation)
  - resource demand (Resource Annotation)

- Are specified by calls to “do nothing” static methods of the Annotation class
Programming Model: Annotations

```java
public class Annotation {
    public static void loopAnnotation (int n) {
    }
    public static void callAnnotation (int n) {
    }
    public static void resourceAnnotation (String ann) {
    }
}
```

```java
public void pippo(int num) {
    int i=0;
    while ( i<num ) {
        Annotation.loopAnnotation(100);
        ......
        i ++;
    }
    ......
    Annotation.callAnnotation(100);
    recursive(num);
}
```

```java
public void recursive(int iter) {
    if(iter==0)
        return;
    System.out.println(iter);
    Recursive(iter-1);
}
```

```java
public void pippo(int num) {
    Annotation.resourceAnnotation(“Energy(10),Memory(100)“);
    motorola.device.getSystemInfo();
    ......
}
```
Analyzer

Analysis basic idea:
- Simulate each possible execution path in the java bytecode
- For each path:
  - Match each encountered bytecode instruction against the resource consumption profile to obtain its resource consumption
  - Sum the instructions’ resource consumption
- The final result is a set of resource demands (resource sets), one for each path

Analyzer: issues

```java
public void pippo2(int x) {
    String str;
    if (x==0)
        str = "is equal to 0";
    else
        str = "is NOT equal to 0";
    System.out.println(str);
}
```

```
0: iload_1
1: ifne 10
4: ldc is equal to 0
6: astore_2
7: goto 13
10: ldc is NOT equal to 0
12: astore_2
13: getstatic java/lang/System.out:Ljava/io/PrintStream;
16: aload_2
17: invokevirtual java/io/PrintStream.println:(Ljava/lang/String;)V
20: return
```
Analyzer: issues

```java
public void pippo(int num) {
    int i = 0;
    while (i < num) {
        Annotation.loopAnnotation(100);
        ...
        i ++;
    }
}
```

Bytecode Abstract Syntax Tree

- **Analyzer issues:**
  - analyze each instruction only once
  - reconstruct the block of bytecode instructions that belongs to the source-level loop body

- **Bytecode Abstract Syntax Tree:** finite, node and arc labeled, directed tree where each internal node represents a Java control flow construct (e.g., WHILE) and the children of that node represent meaningful components of the construct (e.g., WHILE condition and body). The role played by a child node in the construct is used to label the arc entering the child node itself. Finally, the leaf nodes represent JVM instructions.
public class Library {
    static Item[] archive = new Item[100];

    public static void print(String title) {
        int i = 0;
        while (i<archive.length) {
            Annotation.loopAnnotation(100);
            if (archive[i].title==title) {
                String s = archive[i].getData();
                Annotation.resourceAnnotation("CPU(20), Bluetooth(true)");
                PrintServices.printBT(s);
                break;
            } else
                i++;
        }
    }
}
Analyzer: Bytecode Abstract Syntax Tree

- How to obtain the BAST of a Java bytecode?
  - Structure Encapsulation Analysis

- DAVA
  - decompiler for arbitrary Java bytecode.

http://www.sable.mcgill.ca/dava/

Analyzer

- The Analyzer has been formalized by using a transition system

- The analysis performs a scanning of the application code by traversing the BAST corresponding to each method, and by incrementally refining the resource demands of the application
Analyzer: transition system

\[ F_1 \quad P_2 \quad \ldots \quad P_n \]
\[ \gamma \rightarrow \gamma' \]

\[ \Gamma: \{\langle e, b, M, n \rangle \} \cup \varnothing(\text{ResourceSet}) \]
\[ T: \varnothing(\text{ResourceSet}) \]

A Java application starts its execution from the main(String[]) method \( \rightarrow \) initial configuration will be \( \langle e, b, \{\text{main(String[])}\}, \text{bast}(e, \text{main(String[])} \rangle \)

Sets of Resource sets sum operation

\[ P_1 \oplus P_2 = \{ p_1 \oplus p_2 \mid \forall p_1 \in P_1, \forall p_2 \in P_2 \} \]

\{ \{\text{CPU(1)}\} \} \oplus \{ \{\text{CPU(2)}\}, \{\text{Energy(1)}\} \} = \{ \{\text{CPU(3)}\}, \{\text{CPU(1)}, \text{Energy(1)}\} \}

```java
void pippo()
{
    if(cond)
    ...
    else
    ...
    if(cond)
    ...
    else
    ...
}
```
Sets of Resource sets scalar multiplication

- Sets of resource sets scalar multiplication operation
  \[ n \otimes P = \{ n \otimes p \mid \forall p \in P \} \]

3 \otimes \{ \{CPU(2)\}, \{Energy(1)\} \} = \{ \{CPU(6)\}, \{Energy(3)\} \}

Fall-Back Leaf Rule

\begin{align*}
\text{IsLeaf}(n) & \quad \text{Label}(n) = \text{instr} \\
\text{instr} & \quad !\text{Like}(\text{"invoke\#"}) \\
!\text{IsAnnotation}(n) & \\
r = b(\text{instr}) & \quad C = \{ r \} \\
\langle e, b, M, n \rangle & \rightarrow_{\lambda A} C
\end{align*}

1) istore_1 \rightarrow \{CPU(2)\}
2) invoke.* \rightarrow \{CPU(4)\}
3) .* \rightarrow \{CPU(1), Energy(1)\}
4) invokestatic LocalDevice.getLocalDevice() \rightarrow \{Bluetooth(true), Energy(20)\}

0: iconst_0 \rightarrow C=\{CPU(1), Energy(1)\}
1: istore_1 \rightarrow C=\{CPU(2), Energy(1)\}
2: iload_1 \rightarrow C=\{CPU(1), Energy(1)\}
10: bipush 100
12: invokestatic Annotation.loopAnnotation(I)V
32: invokevirtual Item.getData()Ljava/lang/String;
36: ldc *CPU(20), Bluetooth(true)*/
38: invokestatic Annotation.resourceAnnotation(…
42: invokevirtual ......
**Annotation Instructions Rule**

\[
\text{IsLeaf}(n) \quad \text{Label}(n) = \text{instr} \\
\text{IsAnnotation}(n) \\
\text{instr} \; \text{Like}(\text{"invokestatic Annotation.resourceAnnotation"}) \\
\left\langle e, b, M, n \right\rangle \rightarrow_{\alpha_1} \emptyset
\]

```
10: bipush 100
12: invokestatic Annotation.loopAnnotation(I)I
```

```
36: ldc "CPU(20), Bluetooth(true)"
38: invokestatic Annotation.resourceAnnotation(......
```

---

**Resource Annotation Rule**

\[
\text{IsLeaf}(n) \quad \text{Label}(n) = \text{instr} \\
\text{IsAnnotation}(n) \\
\text{instr} \; \text{Like}(\text{"invokestatic Annotation.resourceAnnotation"}) \\
\text{r = ResourceAnnotation}(n) \\
\left\langle e, b, M, n \right\rangle \rightarrow_{\alpha_1} \{r\}
\]

```
36: ldc "CPU(20), Bluetooth(true)"
38: invokestatic Annotation.resourceAnnotation(...... \rightarrow \{ \text{CPU(20), Bluetooth(true)} \})
```

---

42

44
Java: dynamic binding and static binding

class Item {
    public String title;
    public String getData() { ... };
    
    class Book extends Item {
        public String getData() {
            String data="Title=" + title;
            return data;
        }
    }
    
    class DVD extends Item {
        public String getData() {
            return "...";
        }
    }
}

Item i;
If(Math.random()<0.5)
i = new Book();
else
    i = new DVD();
i.getData();

Java: dynamic binding and static binding

class Item {
    ... public static void init {...}
    private String aux(){
        return "xxx";
    }
    public void location () {
        String s = aux();
        System.out.println(s);
    }
}

class Book extends Item {
    ... public static void init {...}
    private String aux(){
        return "yyy";
    }
}

Item i = new Book();
Item.init();
...
How JVM handles method invocation

- Static binding
  - `invokestatic` → static methods
  - `invokespecial`:
    - invocation of instance initialization (`<init>`) methods
    - invocation of private methods
    - invocation of methods using the super keyword

- Dynamic binding
  - `invokevirtual` → instance methods

---

Invokestatic & invokespecial rule

\[
\begin{align*}
& IsLeaf(n) \quad Label(n) = instr \\
& instr = invokestatic methId \lor \ instr = invokespecial methId \\
& \text{methId} \notin M \quad instr !Like ("invokestatic Annotation") \\
& r = b(instr) \\
& Bаст(e, methId) = n' \\
& \langle e, b, M \cup \text{methId}, n' \rangle \xrightarrow{AA} C \\
& t = CallAnnotation(n) \\
& C' = \{ r \} \oplus (t \circ C) \\
& \langle e, b, M, n \rangle \xrightarrow{AA} C' \\
\end{align*}
\]

42: `invokestatic PrintServices.printBT(Ljava/lang/String;)V`

`PrintServices.printBT` is a library method → `n'=null → C=∅`

`C'=(r) = \{ (CPU(4), Energy(1)) \}`
Invokevirtual rule

IsLeaf(n)  Label(n) = instr

\[ r = b(\text{instr}) \quad t = \text{CallAnnotation}(n) \]

\[ \text{LookupOverrides}(e, \text{methId}) = S \]

\[ Z = \{ z_1 \ldots z_k \} = \{ s_i | s_i \in S, s_i \notin M \} \]

\[ \forall z_i \in Z \quad \text{Bast}(e, z_i) = n_i \]

\[ \forall z_i \in Z \quad \langle e, b, s, M \cup z_i, n_i \rangle \rightarrow_{AA} C_i \]

\[ \forall i = 1 \ldots k \quad C'_i = \{ r \} \oplus (t \odot C_i) \]

\[ C = \bigcup_{i=1}^{k} C'_i \]

\[ \langle e, b, M, n \rangle \rightarrow_{AA} C \]

---

public class Item {
    public String title;
    public abstract String getData();
}

class Book extends Item {
    public String getData() {
        String data="Title=" + title;
        return data;
    }
}

class DVD extends Item {
    public String getData() {
        return "...";
    }
}

public class Library {
    static Item[] archive = new Item[100];
    public static void print(String title) {
        int i=0;
        while (i<archive.length) {
            Annotation.loopAnnotation(100);
            if (archive[i].title==title) {
                String s=archive[i].getData();
                Annotation.resourceAnnotation(
                    "CPU(20)," + "Bluetooth(true)");
                PrintServices.printBT(s);
                break;
                break;
            } else
                ++i;
        }
    }
}
Invokevirtual rule

32: invokevirtual Item.getData()Ljava/lang/String;

LookUpOverride(e, Item.getData()) = (Book.getData(), DVD.getData())

\[ t = \{CPU(4), Energy(1)\} \]
\[ t_{\text{inv}} = 1 \]

\[ C_{\text{DVD.getData}} = \{\{CPU(2), Energy(2)\}\} \]
\[ C_{\text{Book.getData}} = \{\{CPU(24), Energy(11)\}\} \]
\[ C_{\text{DVD.getData}} = \{\{CPU(6), Energy(3)\}\} \]
\[ C_{\text{Book.getData}} = \{\{CPU(28), Energy(13)\}\} \]
\[ C = \{ \]
\[ \{CPU(6), Energy(3)\}, \]
\[ \{CPU(28), Energy(13)\} \}
\]

\[ IsLeaf(n) \quad Label(n) = \text{instr} \]
\[ \text{instr} = \text{invokevirtual method} \]
\[ t_{\text{inv}} = \text{CallAnnotation}(n) \]
\[ \text{LookupOverrides}(e, \text{method}) = S \]
\[ Z = \{z_{1} \ldots z_{n}\} = \{z_{1} \in S, s_{i} \in M\} \]
\[ \forall z_{i} \in Z \quad \text{Base}(e, z_{i}) = n_{i} \]
\[ \forall s_{i} \in Z \quad \langle e, s_{i}, \text{method}, n_{i} \rangle \rightarrow_{AA} C_{i} \]
\[ \forall n = 1 \ldots K \quad C_{F} = \langle r \rangle + (t_{\text{inv}} \odot C_{i}) \]
\[ C = \bigcup_{i=1}^{K} C_{i} \]
\[ \langle e, b, m, n \rangle \rightarrow_{AA} C \]

class DVD extends Item {

public String getData() {

\[ \text{String data} = \text{"Title} = \text{title}" \]
\[ \text{return data; } \]
\]

\[ \{\{CPU(1), Energy(1)\}\} \]

\[ \{\{CPU(1), Energy(1)\}\} \]

\[ \{\{CPU(2), Energy(13)\}, \{CPU(28), Energy(13)\}\} \]
**BLOCK rule**

Label(n) = BLOCK

Children(n, element) = Z = \{z_1 \ldots z_k\}

∀z_i ∈ Z \hspace{1em} \langle e, b, M, z_i \rangle \xrightarrow{A_A} C_i

C = \bigoplus_{i=1}^k C_i

\langle e, b, M, n \rangle \xrightarrow{A_A} C

{CPU(1), Energy(1)}
IF_ELSE rule

\[
\begin{align*}
Label(n) &= \text{IF\_ELSE} \\
\text{Children}(n, \text{condition}) &= \{n_{\text{cond}}\} \\
\text{Children}(n, \text{trueBranch}) &= \{n_{\text{true}}\} \\
\text{Children}(n, \text{falseBranch}) &= \{n_{\text{false}}\} \\
(e, b, M, n_{\text{cond}}) &\xrightarrow{AA} C_{\text{cond}} \\
(e, b, M, n_{\text{true}}) &\xrightarrow{AA} C_{\text{true}} \\
(e, b, M, n_{\text{false}}) &\xrightarrow{AA} C_{\text{false}} \\
C_1 &= C_{\text{cond}} \oplus C_{\text{true}} \\
C_2 &= C_{\text{cond}} \oplus C_{\text{false}} \\
C &= C_1 \cup C_2 \\
(e, b, M, n) &\xrightarrow{AA} C
\end{align*}
\]
\[ C_1 = \{ \{ \text{CPU}(58), \text{Energy}(20), \text{Bluetooth}(true) \}, \{ \text{CPU}(36), \text{Energy}(10), \text{Bluetooth}(true) \} \} \]

\[ C_2 = \{ \{ \text{CPU}(64), \text{Energy}(26), \text{Bluetooth}(true) \}, \{ \text{CPU}(42), \text{Energy}(16), \text{Bluetooth}(true) \} \} \]
WHILE rule

\[ \text{Label}(n) = \text{WHILE} \]
\[ \text{Children}(n, \text{condition}) = \{n_{\text{cond}}\} \]
\[ \text{Children}(n, \text{body}) = \{n_{\text{body}}\} \]
\[ t = \text{LoopAnnotation}(n_{\text{body}}) \]
\[ (e, b, M, n_{\text{cond}}) \rightarrow AA C_{\text{cond}} \]
\[ (e, b, M, n_{\text{body}}) \rightarrow AA C_{\text{body}} \]
\[ C_{\text{cond'}} = (t + 1) \odot C_{\text{cond}} \]
\[ C_{\text{body'}} = t \odot C_{\text{body}} \]
\[ C = C_{\text{cond'}} \oplus C_{\text{body'}} \]
\[ (e, b, M, n) \rightarrow AA C \]
The Analyzer

**PROBLEM:** The analyzer is computationally expensive

The Analyzer

Long waiting time to deliver the best application alternative
The Analyzer

PROPERTY: instructions demand is context independent

SOLUTION: split the Analyzer

Abstract Resource Calculator

Abstract Resource Analyzer

Abstract Code Analyzer

- abstracts the JVM
- Matches Resource profile with the occurrences of bytecode patterns and
- Derives the resource demand

Application bytecode characterization

Abstract Resource Calculator

Resource profile

Demanded resources
Abstract Resource Analyzer

1) `aload_0` {CPU(2)}
2) `invokestatic` 
   `LocalDevice.getLocalDevice()` → 
   `{Bluetooth(true), Energy(20)}`
3) → {CPU(1), Energy(1)}

Abstract Code Analyzer

Resource Demand
{ {Energy(26), CPU(8), Bluetooth(true)}, 
{Energy(8), CPU(10)} }

Abstract Resource Calculator

- Computationally expensive
- Executed “only once”

- Computationally “light”
- Executed at each device request
Customizer

- Determines
  - the best application alternative w.r.t. the execution context
  - the evolution policy

- Bases on:
  - Compatibility
  - Goodness
Customizer: compatibility
- Is used to decide if an application can run safely on a device
- A resource demand is compatible with a resource supply if the execution environment supplies a “sufficient amount” of every resource demanded by the alternative.

```
Demand_{GPRS} = \{ \text{GPRS(true), GPRSNet(true)} \}
Demand_{WiFi} = \{ \text{WiFi(true), WiFiNet(true), Battery(high)} \}
Supply_2 = \{ \text{GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(true), Battery(high)} \}
Supply_1 = \{ \text{GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(false), Battery(low)} \}
```

Customizer: goodness
- Is used for choosing the best compatible application alternative w.r.t. a given execution context
- Is based on a notion of priority ($P$) among resources

```
Demand_{GPRS} = \{ \text{GPRS(true), GPRSNet(true)} \}
Demand_{WiFi} = \{ \text{WiFi(true), WiFiNet(true), Battery(high)} \}
Supply_2 = \{ \text{GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(true), Battery(high)} \}
Supply_1 = \{ \text{GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(false), Battery(low)} \}
```

The goodness is calculated as:

$$\text{goodness} = \sum_{r \in \text{sources}} P(r) \cdot \frac{\text{Demand}(r)}{\text{Supply}(r)}$$
Customizer: evolution policy

Determines the best alternatives to deploy if changes occur in the resource supply

\[
\text{Demand}_{\text{GPRS}} = \{ \text{GPRS(true), GPRSNet(true)} \}
\]

\[
\text{Demand}_{\text{WiFi}} = \{ \text{WiFi(true), WiFiNet(true), Battery(high)} \}
\]

\[
\text{Supply}_2 = \{ \text{GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(true), Battery(low)} \}
\]

IF \( \text{Supply} = \{ \text{GPRS(true), GPRSNet(true), Battery(low)} \) \n
SWITCH TO GPRS

---

CHAMELEON Development Environment

alternative class WiFi adapts e-learningMidlet {
void connect() {
  plasticMiddleware.connectViaWiFi();
}
void getLesson() { /* stream the video lesson with its multimedia contents*/
  Annotation.resourceAnnotation("Battery(high)");
  ...}
void saveState() { /* save the current frame of the video less.*/
  ...}
void restoreState() { /*restore the saved state*/
  ...}
}

alternative class GPRS adapts e-learningMidlet {
void connect() {
  plasticMiddleware.connectViaGPRS();
}
void getLesson() { /* stream the lesson slides */
  ...}
void saveState() { /* save the number of the current slide */
  ...}
void restoreState() { /*restore the saved state*/
  ...}
}

---

adaptable public class e-learningMidlet extends PMidlet {
State state;
public e-learningMidlet() {...}

/*life-cycle management methods*/
void pauseApp() {...}
void startApp() {...}
void destroyApp() {...}

/* CHAMELEON specific state management methods */
adaptable void saveState();
adaptable void restoreState();

/* e-learning specific methods */
adaptable void connect();
adaptable void getLesson();
...}
e-learning scenario

Demand_{GPRS} = [ GPRS(true), GPRSNet(true) ]
Demand_{WiFi} = [ WiFi(true), WiFiNet(true), Battery(high) ]

P(GPRS)=1, P(Wifi)=2, P(GPRSNet)=1, P(WifiNet)=2, P(Battery)=1

Supply_{i} = [ GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(false), Battery(low) ]

- GPRS alternative is deployed via OTA
- Evolution policy = if (Supply = [ WiFi(true), WiFiNet(true), Battery(high) ]) switch to WiFi Alternative

 Supply = [ GPRS(true), WiFi(true), GPRSNet(true), WiFiNet(true), Battery(high) ]
 Evolution policy = if (Supply = [ WiFi(true), WiFiNet(true), Battery(high) ]) switch to WiFi Alternative

GPRS Alternative saveState is invoked
GPRS Alternative is un-deployed
WiFi Alternative is deployed
WiFi Alternative restoreState is invoked
The CHAMELEON Framework: considerations

Ongoing work
- Real case-study validation
  - does Chameleon deliver the “best” adaptation alternative?
- Extend the programming model
- Optimize the analyzer
- Improve the Eclipse plug-in
- Use CHAMELEON to obtain adaptable services
Future Work

- Investigate new goodness & priority functions
- Investigate possible relationships between resources’ consumption
  - Energy = 2 · CPU
  - derive new resource demand without performing again the analysis: CPU(10) \rightarrow Energy (20)
  - Energy = Bluetooth(true) ? 4 · CPU : 2 · CPU
- Make a compromise between self-contained and tailored adaptable applications based on:
  - knowledge of user mobility patterns
  - deploy a self-contained application that embeds all the alternatives suitable for the execution contexts in the mobility pattern

References


http://www.di.univaq.it/chameleon/
Conclusion

- An approach to resource-aware application adaptation

- Adaptation at deployment time using a tailored approach

- Evolution by dynamically un-deploying the no longer apt application alternative and subsequently (re-)deploying a new alternative with the desired aptitude.

Thanks!